AFRL-AFOSR-VA-TR-2016-0227

A Logical Framework for Service Migration Based Survivability

Yanjun Zuo
University of North Dakota

06/24/2016
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15-06-2016	Final Performance Report	June 2012 - May 2016

4. TITLE AND SUBTITLE A Logical Framework for Service Migration Based Survivability	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER FA9550-12-1-0131
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Zuo, Yanjun	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of North Dakota 293 Centennial Drive Stop 8363 Grand Forks, ND 58202	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U. S. Air Force Office of Scientific Research 875 N. Randolph Street Arlington, VA 22203	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFOSR
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research aims at developing a logical framework for service migration based survivability in which service migration is an effective mechanism to dynamically transfer critical services from compromised platforms to other clean platforms to ensure that the critical functions will be continuously provided. The research methodology involves defining logic models, system properties, and service migration decision models. We studied the fundamental ingredients and characteristics of a service migration and the key system properties that support an assured service migration. A formal logic was developed for service migration modeling, survivability policy specification, and system property verification. A holistic approach was developed for service migration decisions, which manage and guide the activities and procedures of a service migration process. The approach includes three major decision components – (1) determining whether a service migration is the most appropriate course of action to take in case of a malicious attack; (2) deciding the best strategy for a service migration; and (3) specifying an effective and efficient schedule for the service migration activities.

15. SUBJECT TERMS

Service migration, survivability, logical framework, modeling, decision making

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		17	Yanjun Zuo
					19b. TELEPHONE NUMBER *(Include area code)* (701) 777-2517

Reset

DISTRIBUTION A: Distribution approved for public release.

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Final Performance Report

Submitted to
U.S. Air Force Office of Scientific Research

By
The University of North Dakota

Proposal Title: A Logical Framework for Service Migration Based Survivability

Grant Number: FA9550-12-1-0131

Principal Investigator: Dr. Yanjun Zuo

Program Manager: Dr. Kathleen Kaplan

1. MAJOR ACCOMPLISHMENTS

The major accomplishments of the project include:

- studying the fundamental ingredients and characteristics of a service migration and the key system properties that support an assured service migration;
- developing a formal logic for service migration modeling, survivability policy specification, and system property verification;
- developing a viable method for linking survivability constraint solving to logic reasoning;
- modeling service migration and studying critical factors that affect an effective service migration;
- specifying service migration semantics and constraint solving;
- developing a belief-based decision approach for determining service migration in case of a security incident;
- constructing a fuzz inference model to identify a service migration strategy;
- developing a logic approach for service migration scheduling;
- developing a mobile agent-based scheme for service migration simulation and verification;
- proposing a constrained, possibilistic logic approach for system survivability evaluation.

2. EXECUTIVE SUMMARY

Information systems have been continuously used in many high security and high integrity settings to support our society's critical services including national defense and homeland security. Any disruption of those systems, even for a short period of time, could result in severe consequences. In order to respond to security incidents and survive devastating attacks, a critical system must be able to adapt to its operating environments dynamically. The approach that we study in this research is to equip the system with an ability to migrate the critical services from the compromised platforms to other clean, healthy platforms. Service migration is an important strategy for system survivability. In a situation where component replication is difficult or damage masking fails, service migration is a viable solution to ensure that the critical services can be continuously provided even in case of malicious attacks. Conceptually, service migration involves suspending the current service state, moving the core service programs and other trustworthy space to other platforms, and resuming where computation was left off on the new platforms. Service migration helps a system avoid further loss in case of a security incident and ensures service continuity even when part of the system has been damaged.

In this research we aim at developing a logical framework of service migration for system survivability. More specifically, we have (1) developed a formal logic to represent and reason about system properties to support an assured service migration, (2) specified a semantic model for service migration, (3) modelled service migration process and studied the critical factors that affect an efficient service migration, and (4) developed a holistic approach for service migration decisions including three decision components – (a) determining whether a service migration is the most appropriate course of action to take in case of a malicious attack; (b) specifying the best strategy for a service migration; and (c) developing an effective and efficient schedule for the service migration activities. In addition, we have developed an agent-based system for service migration simulation and a constrained, possibilistic logic for system survivability evaluation.

2.1 A Logical Framework for Service Migration

In developing a formal logical framework to represent and reason about system properties to support an assured service migration, we first specify an abstract system model which lays out the architectural foundation to model various components of a service migration and relocation. Stripping details of a system and its properties to their necessity and applying formal analysis allow us to study the survivability strength and criticality of the system components and their functional/security properties for an assured service migration. In the system architecture, a migration-enabled system (denoted as SYS) has the following components and processes:

- $PM = \{p_1, p_2, ..., p_m\}$: a set of distributed computing platforms, where each p_i can support a set of services as represented by its capability set $Abi(p_i)$;
- $SV = \{s_1, s_2, ..., s_n\}$: a set of services supported by the platforms of SYS. The notation $Ex(s_j, p_i)$ is used to represent that a service s_j is executed on platform p_i during a particular time period;
- *Migration Manager* (*MM*): A component of SYS which coordinates the service migration activities. As part of *MM*, a scheduler executes a function $Choose(p_i, s_j, p_k)$ to generate a service migration arrangement for each service $s_j \in SV$ to be migrated from its current platform p_i to a new healthy platform $p_k \in PM$. Service migration is necessary in case of a detection of severe damage to the platform p_i, or the current platform cannot satisfy the security requirements of the services given the changed operating environment;
- $M(s_j, p_i, p_k) \equiv Ex(s_j, p_i) \Rightarrow Ex(s_j, p_k)$: a service migration process that suspends the current service s_j on p_i, moves its core programs (and other trustworthy space) to a new platform p_k, and resumes where service was left off on the new platform;
- $R(s_j, p_k, p_i) \equiv Ex(s_j, p_k) \Rightarrow Ex(s_j, p_i)$: a service relocation process in which the service s_j is transferred from the migrated platform p_k to its original platform p_i after p_i has been recovered from the damage and the operating environment has improved.

We have specified the major activities and the timeline of a service migration/relocation process as shown in Figure 1. The entire process is triggered by an event such as the detection of severe damage to platform p_i. The first step is for the scheduler to generate a feasible arrangement for each critical service s_j currently executed on p_i to be migrated to a healthy platform p_k. In the meantime, s_j is halted, e.g., freezing service processes, recording global data (service configuration and state), recording the states of individual processes, and terminating the entire service program. The migration process $M(s_j, p_i, p_k)$ starts immediately when the alternative service platform is determined and s_j is appropriately halted. $M(s_j, p_i, p_k)$ is composed of three sub-actions: (1) migration preparation; (2) service/data transfer; and (3) service setup on p_k. The service s_j may be executed on the new platform p_k until its completion. For a long-running service, however, if the previously damaged platform p_i has been recovered, a relocation process $R(s_j, p_k, p_i)$ may transfer s_j back to p_i where it can be completed. $R(s_j, p_k, p_i)$ is composed of three sub-actions similar to $M(s_j, p_i, p_k)$.

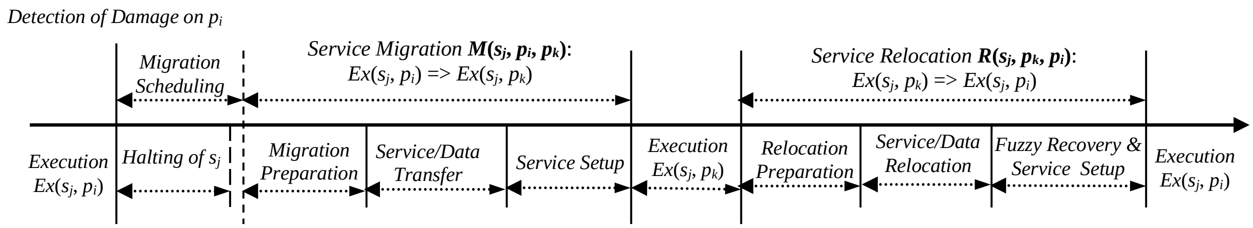

Fig. 1: Migration and Relocation Actions/Events and Timeline for Service s_j

To ensure those migration activities are carried out correctly, we must specify the requirements for the key system and service properties, which, if satisfied, will provide an assured service migration. A formal logic has been developed for system activity specification in which important service characteristics are preserved during and after a migration. Our logical framework provides means to represent and verify that a system with the required properties satisfies a user's policy in terms of the desired survivability objectives. We specify a set of requirements on system/service properties as domain specific constraints. Logic reasoning and constraint solving are separately designed but integrated through the applications of the logic inference rules.

2.1.1 The Logic

In our logic, time is represented by points on the real line and durations are intervals on the real line. Causality among sequences of activities is captured through implications between formulas. The basic types of the logic language include entities (e.g., a platform p_i and a service s_j), actions (*Act*s), events (*Evt*s), time points, time intervals, and various system properties. As a schedulable work, an action represents an activity or a set of activities in a service migration/relocation process. An event is defined as a temporal marker which occurs at a certain time point. For each action *Act*, two time points are implicitly defined, denoted as $Act\uparrow$ and $Act\downarrow$, which represent the starting and ending time points of *Act*, respectively. We use *@Evt* to represent the time point when an event *Evt* occurs and *Du*(*Act*) the duration of the action *Act*.

The formulas and connectives to form the logic are presented next, where *P* represents an atomic formula over an action or an event; *A* and *B* represent atomic or compound formulas; *v* is a variable with a sort (type) *t*; *C* represents a constraint; and *Pro* represents system/service properties that must hold during and after a service migration. If a formula represents an action/event, the formula is true if the represented action/event is (or can be) successfully finished.

(Formulas)	A, B	::=	$P \mid A \wedge B \mid A \vee B \mid A \rightarrow B \mid A \rightarrow_{\Box} B \mid A \rightarrow_{\circ} B \mid A \# B \mid A \ddagger B \mid A \ll_c C \mid C \gg_c A \mid \forall v{:}t.\ A \mid \exists v{:}t.\ A$
(Atomic formulas)	P	::=	$Act \mid Evt$
(Actions)	Act	::=	$Execution \mid Scheduling \mid Halting \mid \ldots$
(Events)	Evt	::=	$Damage\_Detection \mid Choose \mid \ldots$
(Constraints)	C	::=	$Exp \mid Exp@Evt \mid Exp{<}T_1, T_2{>}$
(Constraint expressions)	Exp	::=	$CT \mid CT_1\ Op\ CT_2$
(Constraint terms)	CT	::=	$Pro \mid T \mid Func \mid Exp$
(Constraint functions)	$Func$	::=	$Du(Ac) \mid Remaining(s_j) \mid \ldots$
(Time points)	T	::=	$c \mid @Evt \mid Act\uparrow \mid Act\downarrow$
(Operators)	Op	::=	$< \mid \leq \mid \geq \mid = \mid \ldots$
(Properties)	Pro	::=	$Healthy(p_i) \mid Service\_level(s_j, p_i) \mid \ldots$
(Entities)	$Platforms$	::=	$p_i \mid p_k \mid \ldots$
(Entities)	$Services$	::=	$s_j \mid \ldots$

The meanings of the logic connectives are presented below and their semantics are formally described by the inference rules.

$A \rightarrow\Box B$: formula A implies formula B to be true in the future, i.e., the action/event represented by formula B will start sometime in the future relative to the time when the action/event represented by formula A finishes;

$A \rightarrow\circ B$: formula A implies formula B to be true in the next state, i.e., the action/event represented by A causes B to occur in the next state relative to the time when the action/event represented by A finishes;

$A \ddagger B$: the actions/events represented by A and B are executed concurrently and start at the same time;

$A \# B$: the actions/events represented by A and B are two sequential sub-components of a compound action/event;

$C \gg_c A$: constraint implication – it introduces a pre-constraint C to formula A;

$A \ll_c C$: constraint conjunction – it asserts the validity of a post-constraint C of formula A.

Our logic is designed to represent a service migration/relocation process and to specify the necessary system properties to support an assured service migration. We describe the actions/events and their inherent relationships as a set of system *transition rules* (*TR*s). Each transition rule describes the required system/service behaviors in a service migration/relocation process. If a system is developed with those transition rules and the necessary properties, users can be certain that the system satisfies the survivability policy. In our research, we have identified ten transition rules and only show first two rules in this report.

***TR*$_1$**: $\forall s_j.\ (DD(s_j, p_i) \rightarrow\circ (P(s_j) > L^*)@DD(s_j, p_i) \gg_c H(s_j) \ddagger S(p_i, s_j))$

Transition rule TR_1 represents the temporal relationships among $DD(s_j, p_i)$, $H(s_j)$ and $S(p_i, s_j)$ as well as a pre-constraint for service halting and migration scheduling – immediately after the execution of a critical service s_j on platform p_i is signaled to stop (i.e., $DD(s_j, p_i)$), a scheduling program (i.e., $S(p_i, s_j)$) is executed in order to identify a healthy platform for s_j to migrate to. In the meantime, s_j is appropriately halted (i.e., $H(s_j)$). Since those two actions are performed concurrently, they are represented by a compound formula $H(s_j)\ddagger S(p_i, s_j)$. The pre-constraint of $H(s_j)\ddagger S(p_i, s_j)$ is that the priority of s_j must be greater than L^* (i.e., $P(s_j) > L^*$).

***TR*$_2$:** $S(p_i, s_j) \rightarrow\circ \exists p_k.\ (C(p_i, s_j, p_k) \rightarrow\circ Sh(p_i, s_j, p_k) \ll_c ((s_j \in Abi(p_k) \wedge SL(s_j, p_k) \geq L)@C(p_i, s_j, p_k) \wedge He(p_k)@C(p_i, s_j, p_k)))$

Transition rule TR_2 indicates that if the *Choose* function (i.e., $C(p_i, s_j, p_k)$) identifies a new platform p_k for service s_j to migrate to as a result of the scheduling process (i.e., $S(p_i, s_j)$), s_j is scheduled to migrate from p_i to p_k (i.e., $Sh(p_i, s_j, p_k)$). The post-constraints of $Sh(p_i, s_j, p_k)$ are: (1) p_k can fulfill the necessary functions of s_j (i.e., $s_j \in Abi(p_k)$); (2) the new platform p_k is healthy at the time of the scheduling decision, i.e., $He(p_k)@C(p_i, s_j, p_k)$; and (3) the service level of s_j on the new platform p_k will be maintained at least at a level of L, i.e., $(SL(s_j, p_k) \geq L)@C(p_i, s_j, p_k)$.

The inference rules in our logic are represented using sequent calculus. We start from a sequent (hypothetical judgment) with the format $\sum; \Psi; \Gamma \;\; => \;\; \beta \backslash Ǵ$, where $\sum$ represents a context specifying the sort (or type) of each term or variable appearing in a formula, Ψ represents a set of assumptions in terms of constraints (called *assumption constraints*), Γ represents a set of hypothetic logic formulas, β represents a conclusion formula, and Ǵ represents a set of constraints to be satisfied or solved (called *goal constraints*). The sequent is interpreted as: given all the variables defined in $\sum$, if the assumption constraints in Ψ and the logic hypothesis in Γ are assumed to be true, then we can prove the logic goal β subject to the satisfaction of all the goal constraints in Ǵ.

The constraint formulas in Ψ represent the required system/service properties such as system support for a service migration (e.g., the pre- and post-constraints of the activities in a migration/relocation process), the service quality level on a platform, and the temporal restrictions on system/service actions and events (e.g., the time bound to complete an activity). $\vec{G}$ is the conjunction of all the constraints that must be satisfied given the constraint assumptions in Ψ. $\vec{G}$ is specified for the entire proof process and hence it is checked in the last step.

Γ includes two categories of hypotheses: (1) the system transition rules which model sequences of service migration/relocation actions; and (2) a set of "known facts" such as a damage detection on a platform at a certain time point (modeled as time zero), an arrangement for a service to migrate to a new platform, and a notification of a compromised platform being repaired. Those formulas are used as assumptions for logic reasoning.

$$\frac{check(\Psi_1,\Psi_2) \quad \Sigma; \Psi_1; \Gamma \Rightarrow A\backslash\vec{G}_1 \quad \Sigma; \Psi_2; \Gamma, B \Rightarrow \varphi\backslash\vec{G}_2}{\Sigma; \Psi_1\cup\Psi_2, \Gamma, A \rightarrow B \Rightarrow \varphi\backslash(\vec{G}_1\cup\vec{G}_2)}\ (\rightarrow L\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma, A \Rightarrow B\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A \rightarrow B)\backslash\vec{G}}\ (\rightarrow R\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma, A, B \Rightarrow \varphi\backslash\vec{G}}{\Sigma; \Psi; \Gamma, A\wedge B \Rightarrow \varphi\backslash\vec{G}}\ (\wedge L\ \text{rule})$$

$$\frac{check(\Psi_1,\Psi_2) \quad \Sigma; \Psi_1; \Gamma \Rightarrow A\backslash\vec{G}_1 \quad \Sigma; \Psi_2; \Gamma \Rightarrow B\backslash\vec{G}_2}{\Sigma; \Psi_1\cup\Psi_2, \Gamma \Rightarrow (A\wedge B)\backslash(\vec{G}_1\cup\vec{G}_2)}\ (\wedge R\ \text{rule})$$

$$\frac{check(\Psi,(A\sim\circ B)) \quad \Sigma; \Psi; \Gamma, A\rightarrow B \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi,(A\sim\circ B)); \Gamma, A \rightarrow\circ B \Rightarrow \varphi\backslash\vec{G}}\ (\rightarrow\circ L\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma \Rightarrow (A \rightarrow B)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A \rightarrow\circ B)\backslash admit(\vec{G},(A\sim\circ B))}\ (\rightarrow\circ R\ \text{rule})$$

$$\frac{check(\Psi,(A\sim\Box B)) \quad \Sigma; \Psi; \Gamma, A\rightarrow B \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi,(A\sim\Box B)); \Gamma, A \rightarrow\Box B \Rightarrow \varphi\backslash\vec{G}}\ (\rightarrow\Box L\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma \Rightarrow (A \rightarrow B)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A \rightarrow\Box B)\backslash admit(\vec{G},(A\sim\Box B))}\ (\rightarrow\Box R\ \text{rule})$$

$$\frac{\Sigma\vdash var(C) \quad \Sigma; \Psi; \Gamma, A \Rightarrow \varphi\backslash\vec{G}}{\Sigma; \Psi; \Gamma, C \gg_c A \Rightarrow \varphi\backslash admit(\vec{G}, C)}\ (\gg_c L\ \text{rule})$$

$$\frac{\Sigma\vdash var(C) \quad check(\Psi, C) \quad \Sigma; \Psi; \Gamma \Rightarrow A\backslash\vec{G}}{\Sigma; admit(\Psi, C); \Gamma \Rightarrow (C \gg_c A)\backslash\vec{G}}\ (\gg_c R\ \text{rule})$$

$$\frac{\Sigma\vdash var(C) \quad check(\Psi, C) \quad \Sigma; \Psi; \Gamma, A \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi, C); \Gamma, A \ll_c C \Rightarrow \varphi\backslash\vec{G}}\ (\ll_c L\ \text{rule})$$

$$\frac{\Sigma\vdash var(C) \quad \Sigma; \Psi; \Gamma \Rightarrow A\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A \ll_c C)\backslash admit(\vec{G}, C)}\ (\ll_c R\ \text{rule})$$

$$\frac{check(\Psi,(A\#B)) \quad \Sigma; \Psi; \Gamma, A\rightarrow\circ B \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi,(A\#B)); \Gamma, A\#B \Rightarrow \varphi\backslash\vec{G}}\ (\#L\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma \Rightarrow (A\rightarrow\circ B)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A\#B)\backslash admit(\vec{G},(A\#B))}\ (\#R\ \text{rule})$$

$$\frac{check(\Psi,(A\ddagger B)) \quad \Sigma; \Psi; \Gamma, A\wedge B \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi,(A\ddagger B)); \Gamma, A\ddagger B \Rightarrow \varphi\backslash\vec{G}}\ (\ddagger L\ \text{rule})$$

$$\frac{\Sigma;\Psi; \Gamma \Rightarrow (A\wedge B)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (A\ddagger B)\backslash admit(\vec{G},(A\ddagger B))}\ (\ddagger R\ \text{rule})$$

$$\frac{\Sigma\vdash v{:}t \quad \Sigma; \Psi; \Gamma, [v/x]A \Rightarrow \varphi\backslash\vec{G}}{\Sigma; \Psi; \Gamma, \forall x{:}t.\ A \Rightarrow \varphi\backslash\vec{G}}\ (\forall L\ \text{rule})$$

$$\frac{\Sigma\vdash v{:}t \quad \Sigma; \Psi; \Gamma \Rightarrow ([v/x]A)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (\forall x{:}t.\ A)\backslash\vec{G}}\ (\forall R\ \text{rule})$$

$$\frac{\Sigma\vdash v{:}t \quad \Sigma; \Psi; \Gamma, [v/x]A \Rightarrow \varphi\backslash\vec{G}}{\Sigma; \Psi; \Gamma, \exists x{:}t.\ A \Rightarrow \varphi\backslash\vec{G}}\ (\exists L\ \text{rule})$$

$$\frac{\Sigma\vdash v{:}t \quad \Sigma; \Psi; \Gamma \Rightarrow ([v/x]A)\backslash\vec{G}}{\Sigma; \Psi; \Gamma \Rightarrow (\exists x{:}t.\ A)\backslash\vec{G}}\ (\exists R\ \text{rule})$$

$$\frac{check(\Psi,(A\uparrow= t_1, A\downarrow= t_2)) \quad \Sigma; \Psi; \Gamma, A \Rightarrow \varphi\backslash\vec{G}}{\Sigma; admit(\Psi,(A\uparrow= t_1, A\downarrow= t_2)); \Gamma, A<t_1, t_2> \Rightarrow \varphi\backslash\vec{G}}\ (<>L\ \text{rule})$$

$$\frac{(x_1, x_2, \ldots x_n) = (y_1, y_2, \ldots y_n)\theta}{\Sigma; \Psi_0; \Gamma, p(x_1,\ldots x_n) \Rightarrow p(y_1,\ldots y_n)\backslash\varnothing}\ (\textit{Initial}\ \text{rule})$$

$$\frac{\Sigma; \Psi; \Gamma \Rightarrow A\backslash\vec{G} \qquad \Psi\vdash\vec{G}}{\Sigma; \Gamma \Rightarrow A}\ (\textit{Constraint Solving}\ \text{rule})$$

Fig. 2: Logic Inference Rules in Sequent Calculus

The inference rules of the logic are represented in Figure 2, where φ represents an arbitrary formula; where $\Sigma\vdash var(C)$ means that the variables in formula C have the appropriate sorts (types) as defined in the context ∑; and where *check*, *admit* and ├ are constraint functions. Basically, *check* verifies whether a constraint is admissible to an assumption constraint set Ψ or whether two constraint sets Ψ_1 and Ψ_2 can be combined. Intuitively, a constraint C is admissible to Ψ if the addition of C to Ψ will not cause any inconsistency between C and the existing constraints. *admit* adds a new constraint to Ψ or $\vec{G}$. $\Psi\vdash\vec{G}$ is to verify whether all the constraints in $\vec{G}$ can be solved given the constraints in Ψ.

The proof process starts with the *Constraint Solving* rule. This rule states that in order to prove a goal formula A without constraint verification (i.e., $\Sigma; \Gamma \Rightarrow A$), we need to show: (1) A is provable with the assumption constraints Ψ and the goal constraints $\vec{G}$ (i.e., $\Sigma; \Psi; \Gamma \Rightarrow A\backslash\vec{G}$), and (2) every goal constraint $C\in\vec{G}$ is solvable given the constraints in Ψ (i.e., $\Psi\vdash\vec{G}$). By solving constraint C, we mean that the constraints in Ψ lead to a resolution that *C* is true. The *Initial* rule states that given the initial known constraints, Ψ_0, we can prove a formula $p(y_1,\dots y_n)$ if $p(x_1,\dots x_n)$ is assumed true and $(y_1,\dots y_n)$ is unified with $(x_1,\dots x_n)$ through the most general unifier θ. There is no goal constraint for the *Initial* rule (i.e., Ǵ $=\varnothing$).

2.1.2 Constraint Solving and Proof Search

To capture the complexity of a service migration and the properties of a system to support an assured service migration, we propose to integrate the domain constraints with logic reasoning to include an efficient constraint solver with such properties as consistent and complete and a decision procedure capable of solving a set of constraints in an effective manner. The logic engine and the constraint solver are separately designed but integrated through the applications of the logic inference rules. This allows one to represent and reason the important features of a service migration implemented by different techniques. Separation of logic and the constraint domain will make the logical framework more modular, scalable, and applicable to a wide range of applications.

The constraint functions verify whether some constraints can be satisfied (i.e., Ψ $\vdash$ Ǵ) or admitted to a constraint set (e.g., *check*(Ψ, *C*), *admit*(Ψ, *C*), *admit*(Ǵ, *C*)). Since constraint checking and solving are driven by a logic engine through the inference rules during a proof process, we have developed an algorithm to explicitly describe the interactions between the constraint manager and the logic engine. The three major tasks as the main components of the algorithm are discussed next.

<u>*Constraint Variable Unification*</u> When a logic inference rule is applied, a logic variable may be unified with a constant or another variable. Since logic variables may appear in the constraint terms in Ψ and Ǵ, the unifier should be propagated to the constraint variables as well. The algorithm maintains the most general verifier and applies it to each constraint term. A basic unifier is determined when the *Initial* rule is applied, i.e., a logic goal formula (e.g., $p(y_1,\dots y_n)$) is unified with a logic assumption (e.g., $p(x_1,\dots x_n)$) in Γ.

<u>*Constraint Checking and Admission*</u> Constraint checking is to verify if a constraint *C* can be admitted to the assumption constraint set Ψ (i.e., *check*(Ψ, *C*)) or two assumption constraint set Ψ_1 and Ψ_2 can be combined without conflict (i.e., *check*(Ψ_1, Ψ_2)). Intuitively, *check*(Ψ, *C*) is to verify if a prospective requirement (represented by *C*) for a system property or a time bound on an action can be assumed in a logic reasoning process for an assured service migration given some existing constraint assumptions. A new constraint *C* is admissible to $\Psi=\{C_1, C_2, \dots, C_n\}$ if there is no conflict in assigning values to the free variables in $C_1, C_2, \dots, C_n$ given the quantitative constraints expressed by *C*. If indeed there is no conflict, *admit*(Ψ, *C*) returns a new assumption constraint set with *C* added. Otherwise, *check*(Ψ, *C*) returns false.

<u>*Constraint Solving*</u> Constraint solving is to solve all the goal constraints Ǵ given the assumption constraints in Ψ (i.e., Ψ $\vdash$ Ǵ). By solving a constraint $C\in$ Ǵ (i.e., Ψ $\vdash$ *C*), we mean to check if (1) every variable in *C* has been resolved to a ground term; and (2) the constraint equation/inequality relationship represented by *C* holds given the existing constraints in Ψ. A constraint manager reduces constraint solving to a multi-criteria linear equations/inequalities checking problem. Constraint solving is the last step in a proof process when the constraint-unverified sub-sequent (e.g., ∑; Ψ; Γ => *A*\Ǵ) has been proved. Constraint solving makes sure that all the goal constraints in Ǵ can be satisfied given the assumption constraints in Ψ.

A survivability policy specifies a user's requirements for the survivability features of a system. In our framework, such a policy is represented in terms of a set of temporal and functional properties that the system must have in order to support an assured service migration. If a system possesses those properties, it essentially guarantees that the critical services can be dynamically reallocated from a compromised platform to other healthy ones. Therefore, the entire system can go through malicious attacks and continuously provide mission-critical services. If that is the case, we say the system satisfies the user's survivability policy. As a generic case, the service migration-based survivability policy can be specified from three aspects as represented by the following logic goal statements:

(1) G_1(***Soundness***): $\exists p.(\rightarrow\Box(D(s_j, p) \ll_c (He(p) \wedge SL(s_j, p) \geq L))$

A service s_j will be eventually completed on a healthy platform p with a service level at least L – either on its originally executed platform p_i or the migrated platform p_k as long as the platform is damage-free (healthy).

(2) G_2 (***Efficiency***): $(\rightarrow\circ(H(s_j)\ddagger S(p_i, s_j) \rightarrow\circ M(s_j, p_i, p_k) \rightarrow\Box \quad R(s_j, p_k, p_i)) \ll_c (Du(H(s_j)\ddagger S(p_i, s_j))+Du(M(s_j, p_i, p_k))+Du(R(s_j, p_k, p_i)) \leq D_{max})$

The total time spent on service scheduling $S(p_i, s_j)$, halting $H(s_j)$, migration $M(s_j, p_i, p_k)$, and relocation $R(s_j, p_k, p_i)$ must not be more than the maximum allowable time D_{max}.

(3) G_3 (***Integrity***): $\forall R(s_j, p_k, p_i). (\exists M(s_j, p_i, p_k). (M(s_j, p_i, p_k) \rightarrow\Box R(s_j, p_k, p_i)))$

For every relocation process $R(s_j, p_k, p_i)$, there must exist a migration process $M(s_j, p_i, p_k)$ that occurred earlier.

To verify that a system satisfies the survivability policy as specified by the above three goal statements, it is only necessary to find a proof for "$\sum; \Gamma => (G_1 \wedge G_2 \wedge G_3)$".

A proof search is to identify a derivation of a goal statement from a list of hypothetical assumptions subject to a set of constraints by applying a set of inference rules. A proof is logically viewed as a tree rooted by the conclusion sequent (e.g., $\sum; \Gamma => (G_1 \wedge G_2 \wedge G_3)$ as shown above), where the leaf sequents are all axioms and each non-leaf sequent is derived from its premise sequents by a rule application. The proof search is syntax-driven by following the logic inference rules. Each application of an inference rule reduces a sequent matching the conclusion of the rule to the premises of the rule (i.e., sub-sequents). A branch of the proof is successfully terminated when the formula to be proved unifies with a formula in the hypothesis set Γ. The resulting unifier is propagated to the next remaining premise (including the constraint formulas in Ψ and Ϛ as we discussed earlier) and the process is repeated. The proof search follows the following rules: (1) if every leaf node is an instance of an axiom, i.e., $\sum; \Psi_0; \Gamma, p(x_1,\dots x_n) => p(y_1,\dots y_n)$, the proof search has terminated successfully; (2) if some leaf contains no logic connectives, but is not an instance of any axiom, then the search has terminated unsuccessfully; and (3) if a leaf contains some logic connectives, a search step may choose one connective and apply the corresponding inference rule to reduce the proof of the conclusion to its premises.

2.2 Service Migration Modeling and Critical Factors that Affect an Effective Service Migration

To quantify the important factors that affect an effective and efficient service migration/relocation, we have developed a simulation model to represent the activities and behaviors of system components in a service migration/relocation process. The model is encoded in the Performance Evaluation Process Algebra (PEPA). PEPA introduces delays and probabilistic occurrences to process algebras. The timing behavior of a system is quantified by associating a random variable with each activity, representing its duration. Behavior uncertainty is determined by probabilistic branching – the probabilities of the occurrence of some activities are determined by a race condition between the enabled activities. In our model, the service migration and relocation activities are represented as stochastic actions that are non-deterministic and whose occurrence or non-occurrence is predicted by one or more random variables (i.e., activity rates). A system SYS is modeled as interactions between the service migration/relocation components (i.e., a migrating scheduler, a platform to support a critical service, and a relocation manager) and the damage recovery components (i.e., a fault diagnosing agent and a damage repairer).

The PEPA model representing SYS is shown in Figure 3. As we can see, the model has 11 processes (components) representing a complete procedure for service s_j to be migrated from a compromised platform p_i to a new platform p_k and finally relocated from p_k to p_i after p_i is recovered. The model also includes the recovery procedure of the compromised component.

$$
\begin{aligned}
&Execution_{i\_j} &&\equiv (monitor\_anomaly, p_1).(alarm, al).Contingency_i + (monitor\_normal, p_2).(executing_{i\_j}, f). Execution_{i\_j};\\
&Contingency_i &&\equiv (investigate\_damaged, (it * p_3)).(recovery\_notify_i, tt).Migration\_Manager +\\
& && \quad (investigate\_self\_contain, (it * p_4)).Execution_{i\_j};\\
&Migration\_Manager &&\equiv (halting_{i\_j}, h).Migration\_Scheduler;\\
&Migration\_Scheduler &&\equiv (schedule\_ok,(st * p_5)).Migration_{i\_k} + (schedule\_failure,(st * p_6)).Migration\_Scheduler;\\
&Migration_{i\_k} &&\equiv (m\_Pre, m_1).(m\_Tr, m_2).(m\_Su, m_3).Execution_{k\_j};\\
&Execution_{k\_j} &&\equiv (recovery\_check\_ok, p_7).(recovered_i, T).Relocation\_Manager +\\
& && \quad (recovery\_check\_pending, p_8).(executing_{k\_j}, f).Execution_{k\_j};\\
&Relocation\_Manager &&\equiv (halting_{k\_j}, h).Relocation_{k\_i};\\
&Relocation_{k\_i} &&\equiv (r\_Pre, r_1).(r\_Tr, r_2).(r\_Su, r_3).Execution_{i\_j};\\
&Recovery\_Manager &&\equiv (recovery\_notify_i, T).Recovery_i;\\
&Recovery_i &&\equiv (diagnose, dt).Repairer;\\
&Repairer &&\equiv (repair\_success, (l * p_9)).(recovered_i, rp).Recovery\_Manager + (repair\_fail, (l * p_{10})).\\
& && \quad Recovery_i;\\
&SYS &&\equiv Execution_{i\_j} \underset{L}{\bowtie} Recovery\_Manager\\
&(L = \{recovery\_notify_i, recovered_i\})
\end{aligned}
$$

Fig. 3: PEPA Model of Service Migration and Relocation

The PEPA model has been solved using the PEPA Eclipse Plug-in software. We have developed a Bayesian network decision model to determine the activity rates used in the model. Several rounds of simulations were conducted for steady-state, utilization, passage-time, throughput, and experimentation analysis, in order to study how important factors influence the effectiveness and efficiency of a service migration/relocation process. Those analyses are summarized below.

Steady-state Probabilities, Local State Utilization, and Activity Throughput For a PEPA simulation execution, the system states corresponding to the underlying continuous Time Markov processes are derived and the probability of the system at each state is generated. Our PEPA model has 33 (global) states. Since the model has two top-level PEPA processes: $Execution_{i\_j}$ and *Recovery_Manager*, a (global) state has two elements, one from each local state of the corresponding top-level processes. Our simulation shows that $Execution_{i\_j}$ has 17 local states and *Recovery_Manager* has 4. The PEPA states with dominating steady-state probabilities are those associated with the two local states $executing_{i\_j}.Execution_{i\_j}$ (0.891) and *Recovery_Manager* (0.981). This has also been observed from our utilization analysis, which shows the long-run utilization of each top-level process of the PEPA model. Since $executing_{i\_j}.Execution_{i\_j}$ represents the normal execution of service s_j on its original platform p_i, maximizing the utilization of this state is the objective of an efficient service migration and relocation. We use *Pro* to represent this utilization rate.

Activity throughput A throughput analysis lists the rate at which actions of the PEPA are performed at steady-state. The two PEPA activities with the highest throughputs are $executing_{i\_j}$ (0.09) and *monitoring_normal* (0.09). The former indicates that service s_j is executing on platform p_i and hence a higher value is more desirable. The latter indicates that the intrusion detection system reports system normal operations in most cases (i.e., no suspicious behaviors are detected). Just as the steady-state analysis focuses on the local state $executing_{i\_j}.Execution_{i\_j}$, the throughput of

$executing_{i\_j}$ represents the desired behavior of s_j on p_i; therefore, it is another metric in which we are interested in the research.

<u>*Experimentations*</u> We run the PEPA model with values for its parameters across desired ranges. The experimentations are to study how system security/functional factors affect the utilization rate of $executing_{i\_j}.Execution_{i\_j}$, i.e., *Pro* and the throughput of $executing_{i\_j}$ as mentioned above. We start with the two factors determined by the security features of the system SYS: (1) the probability of anomaly detection on a platform p_i in SYS, i.e., Pro_1 in an intrusion-detection report cycle; and (2) the probability that the detected damage is severe, i.e., Pro_3. Intuitively, if a system has strong security mechanisms and a high level of capability to contain and mask potential damage, then the need for the critical services to be migrated from their normal executing platforms would be low. Hence, the utilization of $executing_{i\_j}.Execution_{i\_j}$ should be higher. The experimentations confirm this observation, i.e., *Pro* decreases when Pro_1 and Pro_3 increase. This clearly indicates that a higher compromise rate on a platform decreases the amount of time that the platform effectively supports the critical services. Furthermore, we have identified that the quantitative relationship between *Pro* and Pro_1 is roughly linear given a fixed Pro_3 rate. This implies that a significant improvement of the system's security will result in an almost equal increase in the normal execution of critical services on their original platforms. A similar pattern can be observed for the throughput of $executing_{i\_j}$ given different Pro_1 and Pro_2 values.

As we have discussed earlier, the *Migration Manager* is responsible for halting a critical service on a compromised platform, scheduling and arranging a new platform for the service to be migrated to, moving the data and program space of the service to the new platform, and finally setting up the service on the new platform. In the meantime, the system component *Recovery Manager* diagnoses the faults and attempts to repair the compromised platform. The performance of those two system components affects a service migration. Our simulation shows that a higher probability of a successful migration-scheduling rate, and a higher probability of a successful repair of a compromised platform, both positively affect the utilization of $executing_{i\_j}.Execution_{i\_j}$, i.e., *Pro*. This indicates that effective damage recovery and highly available healthy platforms increase the overall efficiency of a service migration, which in turn increases the percentage of time that the critical services are executed on their normal platforms. However, *Pro* becomes stable once the possibility of a new platform being identified at the time of migration scheduling reaches a certain value (0.1 in our simulation). That means that any further improvement of migration scheduling beyond this point will no longer significantly improve *Pro*. Therefore, the migration scheduling is not a significant bottleneck for $executing_{i\_j}.Execution_{i\_j}$ beyond that point.

2.3 Semantic Specification of Service Migration

We have developed a semantic model to formalize a service migration, its main constructs, and the constraint rules on the operations and interactions of the service migration activities. We defined the basic notations that describe the core constructs of a service migration, which form the baseline to specify the semantic constraints on the activities of a service migration. The service migration constraints specify what system activities are valid and what properties must hold during and after a service migration. The constraints are defined in terms of (a) the inherent relationships and interactions among system/service activities (e.g., service dependency, resource provision and requirement) and (b) functional and policy regulations on service migration activities (such as service prioritization and platform restrictions). The model serves as a foundation for users to specify and validate the integrity and important properties of a service migration. Some of the basic service migration constrain rules are listed in Table 1.

Table 1: Basic Service Migration Constraint Rules

Constraint Rule	Representation	Explanation
Destination uniqueness	$\forall s_i \in S, \forall p_j \in P, \forall p_t \in P\ (SP(s_i, p_j) \wedge SP(s_i, p_t) \rightarrow \bot)\quad (j \neq t)$	Every service s_i will eventually be migrated to no more than one platform p_j or p_t
Service migration prohibition	$\forall s_i \in S, \forall p_j \in P\ (Prh(s_i, p_j) \rightarrow \neg SP(s_i, p_j))$	A service is specified to be prohibited to execute on a platform due to corporate policy (e.g., security regulations), resource restriction on the platform, service prioritization, and/or technical specifications
Service dependency constraint	$\forall s_i \in S, \forall S' \in 2^S\ (Dep(s_i, S') \rightarrow (\exists s_t \in S', \exists p_j \in P\ (SP(s_t, p_j) \rightarrow SP(s_i, p_j))))$	If one service $s_i \in S$ is dependent on any one service in a group of services, then s_i must be migrated to the same platform p_j as the particular service being depended
Service atomic constraint	$\forall S' \in 2^S\ (Atom(S') \rightarrow (\forall s_i \in S', \forall s_t \in S', \exists p_j \in P\ (SP(s_i, p_j) \leftrightarrow SP(s_t, p_j))))$	If a set of services are mutually dependent on each other (i.e., atomic), then all of them should be migrated to the same platform
Service exclusion constraint	$\forall s_i \in S, \forall S' \in 2^S\ (Exc(s_i, S') \rightarrow (\forall s_t \in S', \exists p_j \in P\ ((SP(s_t, p_j) \wedge SP(s_j, p_j) \rightarrow \bot)))$	If a service $s_i \in S$ is exclusive to every service in a group of services, then s_i must not be executed on the same platform with any one of those services. One service is exclusive to another if they have functional/resource conflicts
Resource provision and requirement constraint	$\forall s_i \in S, \forall p_j \in P\ \ (SP(s_i, p_j) \rightarrow (\forall r_a \in Res\ (RP(p_j, r_a) \geq RR(s_i, r_a))))$	If a service $s_i \in S$ is migrated to a platform, the available resource r_a provided by the platform must be no less than that required by s_i for r_a
Platform health constraint	$\forall s_i \in S, \forall p_j \in P\ \ (SP(s_i, p_j) \rightarrow TH(p_j))$	Each service must be arranged to be migrated to a healthy platform

Based on the semantic model, we have proposed an approach to identify an optimal service migration arrangement with the lowest cost to migrate each service to a platform without violating any of the semantic constraints. Essentially, identifying an optimal service migration arrangement is reduced to a Pseudo-Boolean Optimization (PBO) problem, which is an extended SAT problem. The idea is to formulate and minimize an objective function $\sum_{i=1, j=1}^{i=n, j=m} x_{i,j} * CT_{i,j}$ subject to a set of pseudo-Boolean constraints, where $x_{i,j} \in \{0, 1\}$ represents whether service s_i is arranged to be migrated to platform p_j and $CT_{i,j} \in \mathbb{Z}$ represents the cost to move s_i to p_j. In a PBO problem, each pseudo-Boolean constraint is either in a pure Boolean Conjunctive Normal Form or in a format of $\sum_{j=1}^{j=m} (a_{i,j} * x_j) \geq b_i$, where $a_{i,j}, b_i \in \mathbb{Z}$ and $x_j \in \{0, 1\}$.

We have developed an algorithm to automatically generate the set of pseudo-Boolean constraints given the service migration constraints in our semantic model. A set of simulations have been conducted to evaluate the feasibility and efficiency of using the proposed PBO approach to identify an optimal service migration arrangement without violating any of the service migration constraint rules. We used *Sat4tj-pb* (www.sat4j.org) in our simulations, which is an open-source Java library and the wining program of the Pseudo Boolean Competition 2012. The simulations demonstrated the feasibility of enforcing the semantic constrains in identifying an optimal service migration arrangement using the PBO approach. For performance evaluation, we studied the impact of the number of services to be migrated and various constraint rules on the execution time required to identify an optimal service migration arrangement.

2.4 A Holistic Approach for Service Migration Decision Making

As a systematic defensive security approach, service migration is a system-wide process and involves multiple components of a system. As the complexity of a system and the attacking techniques continuously grow, a well-planned and ensured service migration is necessary in order to minimize any damage resulted from malicious attacks. We have developed a holistic approach for service migration decisions, which manage and guide the activities and procedures of a service migration process. Our approach includes three major decision components – (1) determining whether a service migration is the most appropriate course of action to take in case of a malicious attack; (2) deciding the best strategy for a service migration; and (3) developing an effective and efficient schedule for the service migration activities.

2.4.1 Belief-based Decision Making for Service Migration Determination

The first fundamental decision for a service migration is to determine whether a service migration is the most appropriate course of action to take in a security incident. This decision is made based on the devastating nature of the attack, the damage already caused by the attack, and system resources available to recover and defend against the attack. Service migration is the most appropriate when the attacking effect is so severe that it is difficult to recover the damaged platforms quickly enough to make the services continuously available to users without a noticeable interruption. In this case, the best strategy is to migrate the services from their compromised platforms to other clean, healthy platforms so that those services can be continuously executed on those new platforms. In this way, any critical services will still be available to users even when some platforms of the system have been compromised.

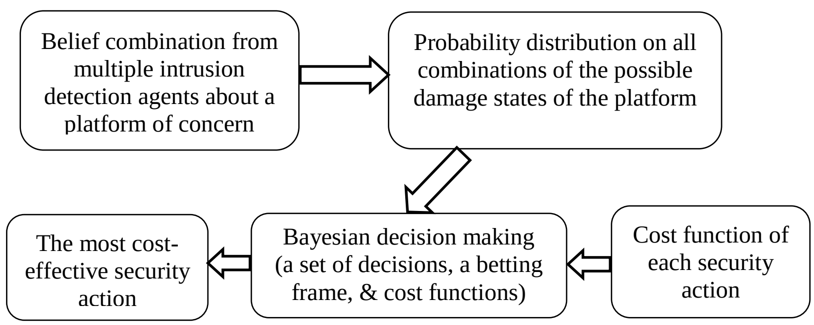

Fig. 4: Belief-based service migration decision model

Making a service migration decision must balance between the cost of service migration (e.g., suspending current running processes, transferring the data and service programs to new platforms, and setting up the services on the new platforms) and the necessity of migrating services somewhere else to avoid further losses (e.g., any direct and indirect costs resulted from the compromised platforms). A fundamental criterion for such a decision is to evaluate whether the platforms of concern have been severely damaged. Assessing the damage status of a platform is not a trivial task given the situation that malicious attacks have become increasingly complicated and system resources available for damage assessment and recovery are often limited in a security incident scenario. Our approach for damage assessment of a platform is to integrate multiple sources of damage assessment results from several independent intrusion detection agents. We have developed a transferable belief-based decision model to represent the damage assessment about a platform as provided by an intrusion detection agent and to combine multiple sources of assessments into an integrated, more reliable damage assessment result about that platform. As shown in Figure 4, damage assessment about a platform by each intrusion detection agent is represented as a basic belief assignment, i.e., a belief mass function on the subsets of a belief domain. Belief combination rules are applied to integrate multiple sources of beliefs to reach a comprehensive belief assignment which represents the final damage assessment of that platform. The combined belief assignment represents a probability distribution on all the combinations of the possible damage states of the platform. Given the cost of performing a security action (e.g., service migration, system repair and restoration, and system mending and refurbishment) on each damage state of the platform, a Bayesian decision model is developed to determine whether a service migration is the most effective and cost efficient action to take. In case the overall cost of service migration is minimum, a decision justifies that service migration is the most appropriate action to take as compared with other security approaches.

2.4.2 Fuzzy Inference for Service Migration Strategy

Once a decision for service migration is made, the next step is to determine which strategy to use for the service migration. In our discussion, a service migration strategy is a specification about whether the service programs, the service state and the data space need to move entirely or partially given different security and system situations. Such a strategy provides a high-level guideline for the underlying service migration activities and procedures to carry out.

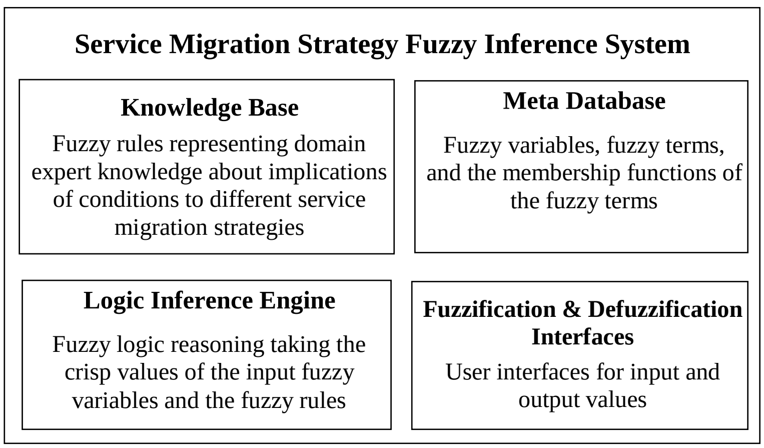

Fig. 5: Components of the Fuzzy Inference System for Determining a Service Migration Strategy

A service migration strategy is determined based on the damage degree of the service programs, the complexity of the service programs, and the availability of network capacity to securely transfer service programs and data to their new platforms. In one situation, if the service programs have been severely damaged, they cannot be executed on the new platforms and therefore should not be moved. Rather, functionally equivalent programs must be generated on the new platforms in order to continuously execute the services. From a security perspective, the newly generated service programs must be resistant to the same type of attacks occurred on the compromised platform. In another situation, if the service programs are only damaged with a minor degree or even damage free, they can be readily used on the new platform and hence can be migrated entirely. Regardless whether the service programs are moved or not, the service state and the data space must be saved and moved to the new platform in order for the services to be resumed from wherever has been left on their original platforms. As a general guideline to determine a service migration strategy, only a minimum amount of data and programs should be migrated whenever possible. We have identified the following three service migration strategies:

- *Heavyweight migration* - moving the entire service programs, the service state, and the data space from their current platform to a new platform;
- *Lightweight migration* - only relocating the service state and the data space but not the service programs. Since the service programs are not moved, the system must re-generate the service code on the new platforms so that the service can be continuously provided on the new platforms;

- *Middleweight migration* - moving part of the service programs, along with the service state and the data space to the new platforms and generating the remaining unmoved service program components on the new platforms in order to execute the service programs.

We have developed a fuzzy inference system to determine a service migration strategy. Our approach uses expert knowledge as linguistic reasoning rules and takes service programs damage assessment, service programs complexity, and available network capability as input. The fuzzy inference system includes four components as shown in Figure 5: (1) a knowledge base containing a set of fuzzy rules that represent domain expert knowledge about the implications of conditions to a service migration strategy. Each rule is represented in linguistic fuzzy terms in a format of If-Then statement, indicating the assumptions and the consequence of a logic implication; (2) a meta database containing fuzzy variables, fuzzy terms, and the membership functions of the fuzzy terms; (3) a logic inference engine for fuzzy logic reasoning taking the crisp values of the input fuzzy variables and the fuzzy rules. In a logic reasoning process, there is no typical "order" for the rules to be applied. The logic engine evaluates the values of the input fuzzy variables and determines the rules that those input values match their conditions. All the matching rules are applied regardless of the order that they appear in the rule base. Methods for condition aggregation, fuzzy rule activation and multi-rule result accumulation are also defined in this component for inference reasoning; and (4) a fuzzification and a defuzzification user interfaces for input and output values. Fuzzification determines the mapping of each input crisp value with the linguistic terms of the fuzzy variable taking this value. Defuzzification converts the fuzzy inference result set to a crisp value for each output variable.

We use jFuzzyLogic to simulate the fuzzy inference system for service migration strategy. jFuzzyLogic is a Java implementation of a fuzzy logic software package, which implements a complete fuzzy inference system as well as fuzzy control logic compliance according to IEC 61131-7 (formerly 1131-7). The definitions of the fuzzy variables of our service migration inference system are encoded in Fuzzy Control Language (FCL). Preliminary results show that the proposed fuzzy inference system is effective in determining the most appropriate strategy for service migration given a security incident scenario.

2.4.3 A Logic Approach for Service Migration Scheduling

One of the important activities of a service migration is service scheduling, which generates an effective arrangement for each service with a high level of priority to migrate from a compromised platform to another healthy one. Before any resources are allocated for service migration and any service migration activities can start, there must be an efficient scheduling to determine which service to be migrated to which platform.

We have proposed a logic approach for service migration scheduling. The logic constructs and inference rules have been developed. Using a logic approach makes it flexible to incorporate new constraint rules and also provides a formal method to analyze, evaluate and verify the correctness of the approach. Given the limited resources available in a security incident and the high requirement for continuous function of the services, service migration scheduling must be completed timely; but in the meantime, the service migration arrangement is subject to various constraints: (1) any new platform to host the migrating services must possess the required capabilities and resources to support the functionality of those services; and (2) the inherent relationships among the migrating services (e.g., dependency or exclusion) on their original platforms must be maintained on the new platforms. In our approach, those requirements are represented as a set of constraints in a logic reasoning process in order to generate a valid service migration schedule. The interplay between the constraint domain and the logic reasoning is implemented through a set of inference rules.

In the implementation, the service migration constraints are enforced by a set of general but expressive rules. Proof obligations and proof restrictions are generated when the corresponding constraint rules are applied. To show that the logic reasoning will not violate any of the constraint rules, a scheduled service must not be prohibited as indicated by a proof restriction. In the meantime, proof obligations are generated as a result of applying some constraint rules. Those proof obligations must be satisfied by some proof elements in order for the entire scheduling process to be successful. A proof solution represents a feasible scheduling of a service on a particular platform, which can be used to discharge some proof obligations. We defined a self-contained data structure to specify a set of constraints, to create proof obligations and proof restrictions when certain constraint rules are applicable, to verify that a reasoning step does not violate the proof restrictions, and to generate proof solutions to solve those proof obligations.

To validate the proposed logic approach for service migration scheduling, we have developed a proof-of-concept logic program to automatically schedule a set of services to be migrated to a set of platforms subject to a set of constraint rules. The logic program takes a set of services and platforms as input and generates an arrangement for each service to be migrated to a platform, or reports a failure if no such arrangement exists or some constraints cannot be satisfied. The program was implemented in JProlog. We run a set of simulations to verify the correctness and efficiency of the program. All the inference rules have been validated. The results show that the logic program has successfully identified all the valid arrangements to migrate the services to available platforms given a set of constraints.

2.4.4 Mobile Agent-based Service Migration Simulation

Mobile agents are special software agents that move spontaneously across multiple hosts of one or more networks. In case of malicious attacks, mobile agents can move from their damaged platforms to other clean, healthy platforms so that the services they offer can be continuously provided on the new platforms, thus achieving service migration. Service migration through such strategic agent movement helps a system survive host damage and improves service availability. We propose a mobile agent-based approach for service migration, where a group of agents collaboratively decide a migration plan to relocate from their current platforms to other more secure and reliable platforms.

We specify the system architecture to support agent migration and propose a collaborative decision making model for a group of agents to decide their destination platforms in a migration process. Since agents are social entities, they collaboratively work with others on certain tasks. Therefore, one agent may functionally depend on other agents. A service migration plan must take this type of dependency into consideration. An algorithm for collaborative migration decision making has been developed, which is executed by a coordinator agent. The algorithm takes as input the local migration decisions $\{S_1, \ldots S_i, \ldots, S_m\}$ from all the m agents in a group and a set of constraint rules R for the agent group. The output is an agent migration plan. Three types of constraint rules are specified in R: (1) atomicity rule – a set of agents must be migrated to the same platform, (2) dependency rule – an agent functionally depends on at least one of subset of agents; therefore, it must migrate to the same platform as the agent which it depends on, and (3) exclusion rule – two or more agents must not be migrated to the same platform. Basically, the collaborative migration decision algorithm recursively takes one platform from each list of feasible platforms provided by one agent, called a tentative migration plan, and then checks this tentative plan against each constraint rule. If any rule is violated, this tentative plan is not feasible and the algorithm moves to the next tentative migration plan until a feasible plan is identified or all the possible tentative plans have been exhausted. A collaborative migration plan makes sure that the agent migration will not violate any of the constraints for the group of agents.

To verify the proposed agent migration scheme, we have developed a proof-of-concept mobile agent system based on Aglets, a Java based agent platform and library for building mobile agent-based applications. An aglet is a Java agent which is specifically designed to support mobility, i.e., allowing an agent to migrate across the hosts of one or more networks. Our simulation includes local and collaborative agent migration decision making as well as the actual agent dispatching to their destination platforms. The result demonstrates the feasibility and efficiency of moving agents from one platform to another using a mobile agent platform.

2.5 A Constrained, Possibilistic Logic Approach for System Survivability Evaluation

We have also developed a logic approach to facilitate users in assessing a software system in terms of the required survivability features. Survivability evaluation is essential in linking foreign software components to an existing system or obtaining software systems from external sources. It is important to make sure that any foreign software components will not compromise the current system's survivability properties. Given the increasing large scope and complexity of modern software systems, there is a need for an evaluation framework to accommodate uncertain, vague, or even ill-known knowledge for a robust evaluation based on multi-dimensional criteria. Our approach incorporates user-defined constrains on survivability requirements. Necessity-based possibilistic uncertainty and user survivability requirement constraints are effectively linked to logic reasoning. A proof-of-concept system has been developed to validate the proposed approach.

In our logical approach for survivability evaluation, the user's survivability requirements are represented in a logic with application specific operators and inference rules. A system's compliance with those requirements are checked through a logic reasoning process. Applying a formal, logic-based approach provides a rigorous verification and guarantee of system properties in a well-structured reasoning process. The design of the logic evaluation framework follows the following principles and guidelines:

(1) Since survivability is a multi-dimensional concept, a software system's properties need to be evaluated from different aspects, including security, adaptability, robustness, and fault tolerance;

(2) Given the increasing scope and complexity of modern software systems, it is virtually impossible for a user to evaluate every property of a system. For an objective and accurate assessment about a system's survivability features, third-party trusted evaluators can be used who are specialized in some particular aspects of system survivability features. Our approach supports collecting survivability property certificates from trusted evaluators, encoded as logic formulas, reasoning on those individual assessments through a logic proof process, and integrating them into a complete survivability evaluation result;

(3) It is often the case that even a specialized evaluator cannot be very certain about a particular feature of a software system. Our approach supports logic reasoning on uncertain, imprecise, or even vague information. This uncertainty-aware reasoning is achieved by defining many-valued logic formulas and necessity-based possibilistic uncertainty, where uncertain information can be formally represented and linked to a logic reasoning process. The proposed approach makes it possible to express fuzzy pattern matching in formal survivability proof;

(4) An evaluation framework should be applicable to practical case scenarios. In terms of users' system property requirements, it should have a mechanism to represent and reason about constraints on the required survivability features of a software system. Some system properties may take others as their pre-requisite conditions. For example, the system's self-healing ability depends on an accurate and timely damage assessment. As another example, the capability of a system to reasonably predict the causes of system faults and take the corresponding corrective actions to recover from damage is closely related to the system's ability to control vulnerability. Therefore,

both of those two properties may be required for the system. Incorporating those and other constraints and allowing an efficient connection between a constraint domain and a logic reasoning process is essential for a survivability evaluation framework to be practical. Our logic framework supports constrained logic reasoning to accommodate these and other types of constraints.

The designed logic supports fuzzy pattern matching for survivability evaluation uncertainty reasoning and user requirement constraint specification and verification. We present a logic mechanism to incorporate survivability requirement constraints and possibilistic uncertainty to software system survivability evaluation. A formal design is presented to link the hybrid worlds of constraint domains to logic reasoning. The interplay between the constraint checking and logic reasoning is supported by a set of logic inference rules. To make sure that the logic inference rules are correct, we have developed a prototyping theorem prover implemented in JProlog. The logic engine is encoded in Prolog. We have conducted a set of experiments for system survivability evaluation. All the logic inference rules have been validated.

3. PERSONNEL SUPPORTED

Faculty:

- Yanjun Zuo (University of North Dakota)

Graduate students (the project supported the following students during their studies at the University of North Dakota):

- Xiwei Wang (M.S., Computer Science)
- Abhilasha Bhatia (M.S., Computer Science)

4. PUBLICATIONS

This project has resulted in the following journal and conference publications as well as a book chapter. Currently, there are still a couple of papers under review, which have not been included in the list below.

- "Reputation-based Service Migration for Moving Target Defense", *Proc. of 2016 IEEE International Conference on Electro/Information Technology*, pp. 239-246, Grand Forks, ND, USA, 2016.
- "A Holistic Approach for Service Migration Decision, Strategy and Scheduling", *Proc. of 10th Annual Symposium on Information Assurance*, pp. 14-18, Albany, NY, USA, 2015.
- "Fuzzy Inference for Service Migration Strategy", *Proc. of IEEE International Conference on Electro/Information Technology*, pp. 54-61, DeKalb, IL, USA, 2015.
- "Mobile Agent-Based Service Migration", *Proc. of 12th International Conference on Information Technology – New Generation*, pp. 8-13, Las Vegas, NV, USA, 2015.
- "Belief-Based Decision Making for Service Migration", *Proc. of 48th Hawaii International Conference on System Science*, pp. 5212-5221, Hawaii, USA, 2015.
- "A Constrained, Possibilistic Logic for System Survivability Evaluation", *Journal of Software Engineering and Knowledge Engineering*, (24)5, pp.777-800, 2014.
- "Semantic Specification of Service Migration for System Survivability", *Proc. of 2014 IEEE International Conference on Electro/Information Technology*, p.6, Milwaukee, WI, USA, 2014.

- "A Logic Based Approach for Service Migration Scheduling", *Proc. of 9th International Conference on Cyber Warfare and Security*, pp. 226-234, West Layfayette, IN, USA, 2014.
- "Data Labeling for Integrity in SCADA Systems", *Proc. of 13th IEEE/ACIS International Conference on Computer and Information Science*, pp. 111-118, Taiyuan, China, 2014.
- "Towards a Service Migration Architecture for Service Availability", *Proc. of 12th International Conference on Security and Management*, pp. 325-331, Las Vegas, NV, USA, 2013.
- "Moving and Relocation: A Logical Framework of Service Migration for Software System Survivability", *Proc. of 7th IEEE International Conference on Software Security and Reliability*, pp. 139-148, Washington D.C., USA, 2013.
- "Modeling Service Migration and Relocation in Mission-Critical Systems", *IFIP WG 11.10 International Conference on Critical Infrastructure Protection, Critical Infrastructure Protection VII, Springer, Heidelberg, Germany*, pp. 155-170, 2013.
- "Composition and Combination-based Object Trust Evaluation for Knowledge Management in Virtual Organizations", *VINE-The Journal of Information and Knowledge Management Systems*, (43)3, pp. 296-321, 2013.
- "Towards a Logical Framework for Migration-based Survivability", *Proc. of 7th Annual Symposium on Information Assurance*, pp. 29-33, Albany, NY, USA, 2012.

1.

1. Report Type

Final Report

Primary Contact E-mail
Contact email if there is a problem with the report.

yzuo@business.und.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

7017776798

Organization / Institution name

University of North Dakota

Grant/Contract Title
The full title of the funded effort.

A Logical Framework for Service Migration Based Survivability

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-12-1-0131

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Yanjun Zuo

Program Manager
The AFOSR Program Manager currently assigned to the award

Kathleen Kaplan

Reporting Period Start Date

06/01/2012

Reporting Period End Date

05/31/2016

Abstract

This research aims at developing a logical framework for service migration based survivability in which service migration is an effective mechanism to dynamically transfer critical services from a compromised platform to other clean platforms in order to ensure that the critical functions will be continuously provided. The research methodology involves defining logic models, system properties, and service migration decision models.
We studied the fundamental ingredients and characteristics of a service migration and the key system properties that support an assured service migration. An abstract system model was specified which lays out the architectural foundation to model various components of a service migration/relocation.

A formal logic was developed for service migration modeling, survivability policy specification, and system property verification. In our logic, time is represented by points on the real line and durations are intervals on the real line. Causality among sequences of activities is captured through implications between formulas. The basic types of the logic language include entities, actions, events, time points, time intervals, and various system properties. The formulas (atomic and compound) and connectives to form the logic are formally defined. The logic is designed to represent a service migration/relocation process and to specify the necessary system properties to support an assured service migration. The actions/events and their

inherent relationships are described by a set of system transition rules. Each transition rule describes the required system/service behaviors in a service migration/relocation process. If a system is developed with those transition rules and the necessary properties, users can be certain that the system satisfies the survivability policy. Basically, the logic provides means to represent and verify that a system with the required properties satisfies a user's policy in terms of the desired survivability objectives. We specified a set of requirements on system/service properties as domain specific constraints. Logic reasoning and constraint solving are separately designed but integrated through the applications of the logic inference rules.

We have developed a holistic approach for service migration decisions, which manage and guide the activities and procedures of a service migration process. The approach includes three major decision components – (1) determining whether a service migration is the most appropriate course of action to take in case of a malicious attack; (2) deciding the best strategy for a service migration; and (3) specifying an effective and efficient schedule for the service migration activities.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

AFD-070820-035.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

Final Performance Report.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

• "Reputation-based Service Migration for Moving Target Defense", Proc. of 2016 IEEE International Conference on Electro/Information Technology, pp. 239-246, Grand Forks, ND, USA, 2016.

• "A Holistic Approach for Service Migration Decision, Strategy and Scheduling", Proc. of 10th Annual Symposium on Information Assurance, pp. 14-18, Albany, NY, USA, 2015.

• "Fuzzy Inference for Service Migration Strategy", Proc. of IEEE International Conference on Electro/Information Technology, pp. 54-61, DeKalb, IL, USA, 2015.

• "Mobile Agent-Based Service Migration", Proc. of 12th International Conference on Information Technology – New Generation, pp. 8-13, Las Vegas, NV, USA, 2015.

• "Belief-Based Decision Making for Service Migration", Proc. of 48th Hawaii International Conference on System Science, pp. 5212-5221, Hawaii, USA, 2015.

• "A Constrained, Possibilistic Logic for System Survivability Evaluation", Journal of Software Engineering and Knowledge Engineering, (24)5, pp.777-800, 2014.

• "Semantic Specification of Service Migration for System Survivability", Proc. of 2014 IEEE International Conference on Electro/Information Technology, p.6, Milwaukee, WI, USA, 2014.

• "A Logic Based Approach for Service Migration Scheduling", Proc. of 9th International Conference on

Cyber Warfare and Security, pp. 226-234, West Layfayette, IN, USA, 2014.

• "Data Labeling for Integrity in SCADA Systems", Proc. of 13th IEEE/ACIS International Conference on Computer and Information Science, pp. 111-118, Taiyuan, China, 2014.

• "Towards a Service Migration Architecture for Service Availability", Proc. of 12th International Conference on Security and Management, pp. 325-331, Las Vegas, NV, USA, 2013.

• "Moving and Relocation: A Logical Framework of Service Migration for Software System Survivability", Proc. of 7th IEEE International Conference on Software Security and Reliability, pp. 139-148, Washington D.C., USA, 2013.

• "Modeling Service Migration and Relocation in Mission-Critical Systems", IFIP WG 11.10 International Conference on Critical Infrastructure Protection, Critical Infrastructure Protection VII, Springer, Heidelberg, Germany, pp. 155-170, 2013.

• "Composition and Combination-based Object Trust Evaluation for Knowledge Management in Virtual Organizations", VINE-The Journal of Information and Knowledge Management Systems, (43)3, pp. 296-321, 2013.

• "Towards a Logical Framework for Migration-based Survivability", Proc. of 7th Annual Symposium on Information Assurance, pp. 29-33, Albany, NY, USA, 2012.

2. New discoveries, inventions, or patent disclosures:

Do you have any discoveries, inventions, or patent disclosures to report for this period?

No

Please describe and include any notable dates

Do you plan to pursue a claim for personal or organizational intellectual property?

Changes in research objectives (if any):

None.

Change in AFOSR Program Manager, if any:

At the beginning of this project, Dr. Robert Herklotz was the program manager. Later on Dr. Kathleen Kaplan was the program manager for this project.

Extensions granted or milestones slipped, if any:

None

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Jun 15, 2016 11:56:57 Success: Email Sent to: yzuo@business.und.edu